Appendix N

Metrics - *The Measure of Success*

Content

N.1 Editors Note

"Metrics: the Measure of Success" © 1994 was originally published as a brochure under the sponsorship of the Hughes Software Network Management Council. Reprinted with Hughes Aircraft Company's permission. All rights reserved.

N.2 Foreword

"Our highest priority operating commitment is to quality and continuous measurable improvement in everything we do."

These words from the statement of Hughes Guiding Values emphasize the importance of quality and continuous measurable improvement. Measurement (using metrics) serves as a powerful management tool for evaluating effectiveness and efficiency. Metrics enable us to manage on the basis of facts and data. Continuous measurable improvement cannot be achieved without measuring an existing process, changing some aspect of the process, and then measuring the result to verify it is an improvement. Measurement is an essential element in supporting Integrated Product Development (IPD). The IPD philosophy employs multi-disciplined teams to integrate and apply the best processes to effectively develop products that satisfy every customer's needs.

Common software processes based on best practices are being implemented throughout Hughes Aircraft Company to gain competitive advantage and reduce risk. These processes include standard reporting practices that define the metrics for monitoring project status and for communicating that information to functional and product management.

The metrics described in this brochure have evolved as best practices from more than 20 years of metrics data collection and reporting. They provide the means to measure cost, schedule, process, product quality, productivity, and technical parameters, each of which contribute to our fundamental measure of success — customer satisfaction. Our challenge is to continuously use data in optimizing our software development processes to ensure that Hughes remains a successful World Class performer in the global marketplace.

Terry R. Snyder
Manager, Software Engineering Division
Chairman, Software Network Management Council
Hughes Aircraft Company

N.3 Introduction

Building a successful business often means building better software. In today's highly competitive markets, where software is increasingly a critical factor, a company's ability to plan and control its software development activities is essential. But, according to the Software Engineering Institute (SEI), most American companies lack a well defined and measurable process for managing software development. Typically, these companies face a high rate of failure because of unpredictable product quality, higher costs, and delivery schedules that are out of control.

There is a better way. Using the SEI's Capability Maturity Model as a framework, we find that implementing process maturity criteria, based on a controlled and measurable software development process, can reduce costly software errors, cut the risk factors, increase productivity and product quality, and shorten cycle time. Those organizations that achieve higher process maturity levels typically demonstrate significant control over their software development processes. One of the keys to achieving this control is the use of metrics. Basically, a metric is a standard of measurement. Just as a yardstick is used to measure height in inches or feet, a software metric allows us to use quantitative values to assess how well we're doing in relation to things such as budget or schedule.

Nowadays, a physician uses a digital thermometer to measure a patient's temperature. If the temperature exceeds the norm by several degrees, it's an indication that something may be wrong. Other tests are run to determine the cause of the increased temperature (e.g., blood pressure, respiration, and white blood count). Once the medical data (usually referred to as the patient's "*vital statistics*") has been collected and analyzed, the doctor can recommend treatment to restore the patient to health.

Metrics have a similar function. During the life of a software development project, metrics are the statistical tools that help the software management organization determine how well it is achieving its scheduled commitments. Basically, metrics provide the factual basis for effectively evaluating a project's performance over time (measured in relation to budget, schedule, and other key factors). Metrics are used by managers to assess the progress being made (the overall "*health*" of the project) or to detect unfavorable trends and do something about them before they become show stoppers. Organizations that wish to improve their software development processes can realize a significant return by establishing metrics programs that include the people, facilities, tools and training required for collecting and interpreting metrics data. Hughes Aircraft Company's expanding emphasis on metrics reflects the company's top-down commitment to quality and continuous measurable improvement. The story that follows provides insight into how metrics can be used to quantitatively — and successfully — manage software projects.

This document is not intended as an academic exercise nor does it cover all of Hughes' metrics activities. It provides a description of 12 metrics that have proven useful in the management of software development projects at Hughes. Our experience with software measurement over the years has shown us that metrics work — they help us do our jobs better. The objective in this brochure is not only to define the metrics themselves, but to describe their use (and ultimate value) in the real world by relating them to a hypothetical project.

N.4 Overview of the Sample Project

This brochure uses a hypothetical software development project to illustrate the uses of metrics. It provides a basis for understanding how metrics actually function in the life and health of a project. Although the data shown is entirely fictitious, the intent is to make this generic software project as "*real*" as possible, based on the sort of problems one would expect to encounter in a development effort of moderate size. The project has the following general characteristics:

- 30 month schedule,
- 82,000 source lines-of-code (SLOC) delivered,
- Staff peak of 33 people,
- Consists of two software builds, and
- At this point in time, the first build has been completed and we are two-thirds into the project schedule.

The story line has one main theme: during the Preliminary Design phase, project personnel tried to reduce the size and complexity of the job through automatically generated code and software reuse. But, this inadvertently created a problem: the new design required more memory than allocated. Without metrics, the problem might not have surfaced until very late in the project, during the final stages of integration and testing (around months 23-24), resulting in a major redesign effort with a severe impact on cost and schedule. With metrics, just as in medical science, the key to success is the early detection of problems.

N.4.1 Project Schedule

Project managers need to visualize how and when key activities are planned to occur over the life of the project. They also need a progress report that tells them where they are in relation to this plan. The Project Schedule is a tool that provides a "*quick picture*" of the project. It maps out the workflow, shows the relationships between activities over time, and illustrates progress. It can also be used to identify problems and as an aid to taking corrective action.

The Project Schedule is structured as a timeline plotting all major activities and milestones from project inception to scheduled completion. This schedule reflects all current officially approved activities, dates, and progress status (including any slippage from the officially approved plan). Also shown (at the bottom of the schedule) is a running total of planned and actual data deliveries for each scheduled timeline increment.

In our sample project, there were some early problems in accomplishing the software design. The need to address these design problems caused delays, which, in turn, had a ripple effect on the design reviews. However, early corrective actions allowed the overall project to recover and get back on schedule.

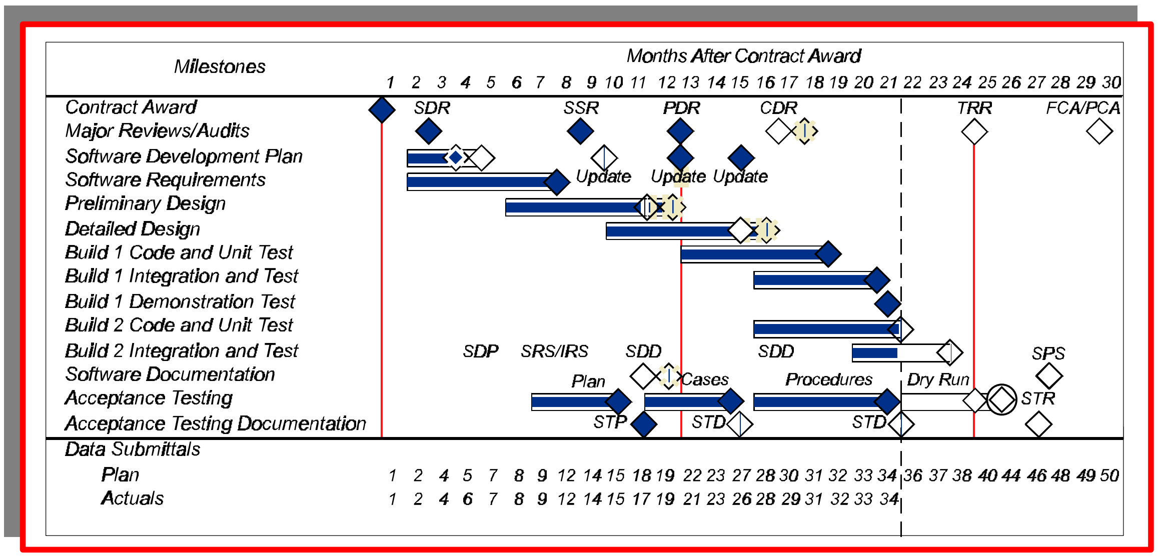


Figure N-1

N.4.2 Milestone Reports

The Milestone Report is intended to identify major software tasks specified in the Project Schedule, intermediate checkpoints for those major tasks, more detailed software component and documentation deliveries, and management-oriented activities (for example, quarterly self-audits). These milestones are listed by description, plan date, latest estimate date, actual date, and reason for slippage.

The Milestone Summary Report is a summary of the data from the Milestone Report that can be merged with milestone data from other projects to measure the organization's overall effectiveness in meeting its customers' expectations for delivery. In order to merge later milestone data from many different projects, a set of standard milestones is included in all project milestone statusing and used as the basis for the data in the Milestone Summary Report.

N.4.3 Rate Chart Report

Software development activities must be planned in great detail. For each of the hundreds (and often thousands) of software units, milestones are established for design, code, unit test, and integration. As time passes, the actual completion dates for these milestones are recorded. This milestone data can be used to measure the software development effort. The reporting of this status is accomplished using Rate Charts.

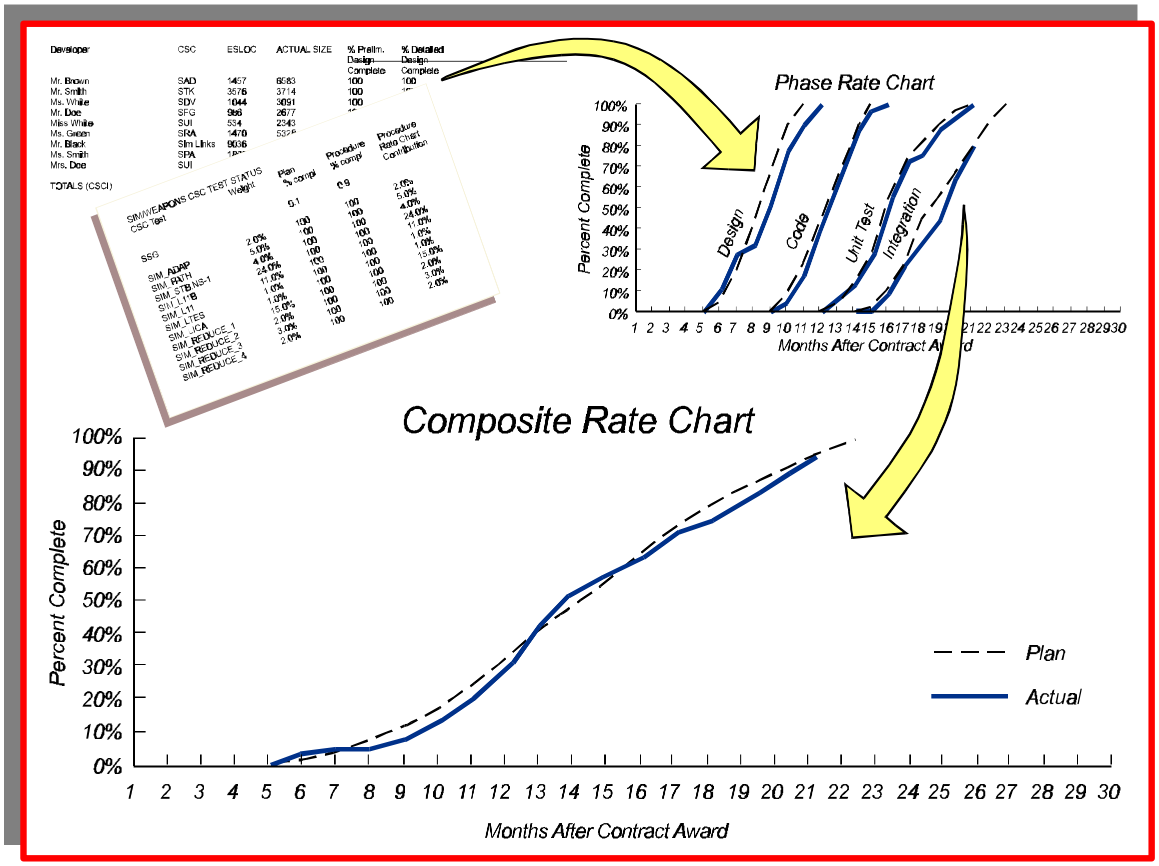


Figure N-2

A Rate Chart for an activity is a two-dimensional graph showing a plan line and an actual line versus calendar time. The plan line shows the percent of milestones planned to be completed versus time. The actual line shows the percent actually completed. With this effective management tool, you can quickly see where you are in relation to the plan at a given point in time, and you can visualize your rate of progress. By comparing the actual line with the planned line, management is alerted to trouble early in the development process and can consider corrective action to remedy the situation (for example, allocation of more resources).

A phase Rate Chart simultaneously provides rate charts for design, code, unit test, and integration. A composite Rate Chart depicts, at a higher level, a weighted summation of several parallel activities of planned and actual work accomplished versus time. It is designed to provide a means to assess the overall progress and status for a project and its individual development activities.

Most software development Rate Charts are defined and measured in terms of software units. However, any activity that can be planned as a detailed set of similar milestones can be depicted as a Rate Chart. Thus, Rate Charts can also be used to report the status of such tasks as documentation and formal test. In our sample project, the Rate Chart shows rapid early progress due, apparently, to the lower complexity of the job. But, once the design problem was caught and corrective action took effect, the rate dropped off and then gradually recovered.

N.4.4 Earned Value Report

The Earned Value Report compares work accomplished against work planned. In this report, the financial budget (Budgeted Cost of Work Scheduled or BCWS) is plotted along with the actual expenditures (Actual Cost of Work Performed or ACWP). A third plot, which represents earned value or the amount of the job that has been completed (Budgeted Cost of Work Performed or BCWP), is then added to the picture. This graphically illustrates the project schedule and financial status at a glance. What you get is an overview of the project's health. For example, if half the money has been spent (ACWP), then half the job should be completed (BCWP). If money is being expended (ACWP) faster than planned (BCWS), there should be a corresponding assessment of progress against the project's milestones (BCWP).

Based on the earned value plots, two additional key indicators can easily be derived and plotted: the Cost Performance Index (CPI) and the Schedule Performance Index (SPI). The CPI is the ratio of work completed in dollars (BCWP) divided by the actual cost of performing the completed work (ACWP). Values greater than "*1*" mean it is costing less than originally planned to perform the work. The SPI is the ratio of work accomplished (BCWP) divided by work planned (BCWS). Numbers greater than "*1*" signify that work is being achieved quicker than originally planned (i.e., the activity is ahead of schedule).

We see in our sample project that, around month 8, there was a critical dip in the key indicators: money was being spent, but very little (in terms of earned value) was being accomplished. We were faced with a slowdown in performance. Immediate corrective action was needed to offset the possibility of a cost overrun and/or late delivery. Both the SPI and the CPI suffered as a result, but the project gradually recovered once the design problem was solved. Easy to use and understand, these key indicators are needed to measure and assess a project's progress and health. They convey powerful messages. Are we meeting our schedule? Are we meeting costs? Is the project healthy?

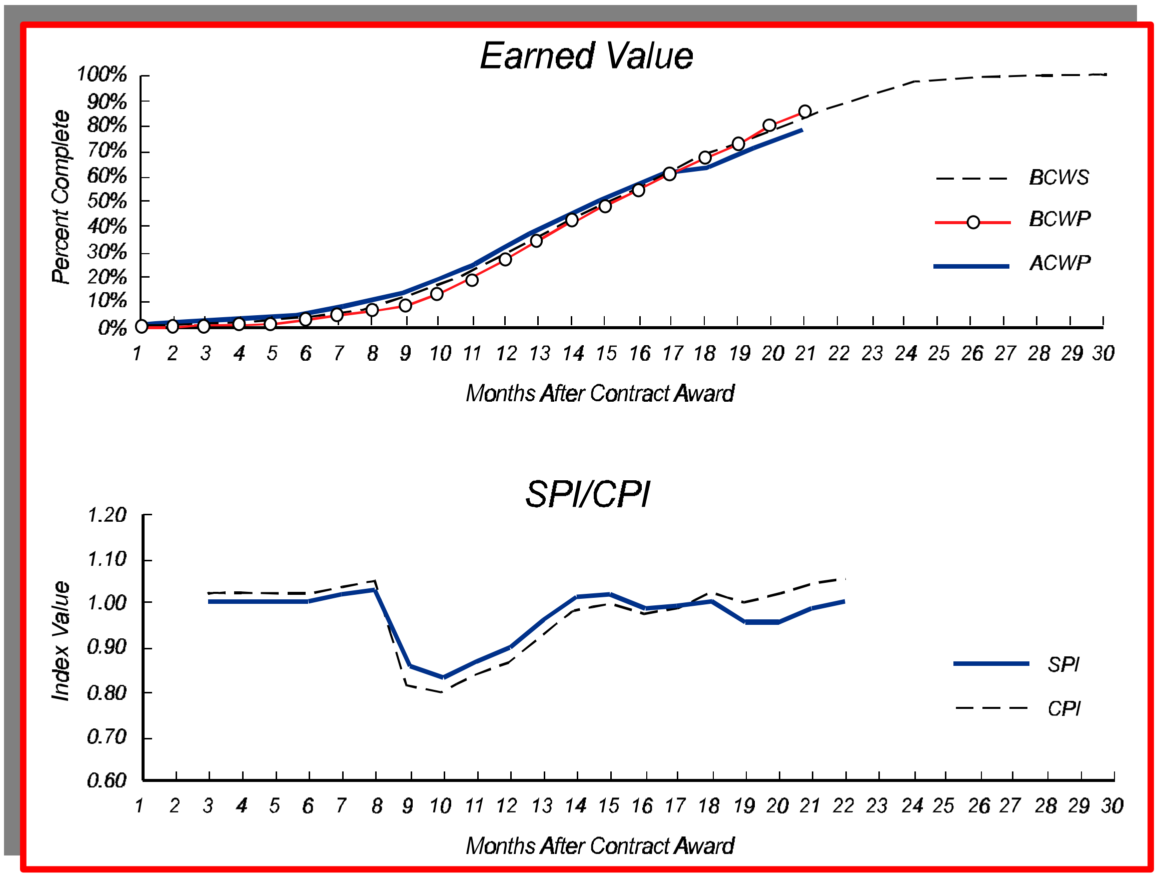


Figure N-3

N.4.5 Financial/Staffing Report

The Financial/Staffing Report provides a clear indication to management of a project's performance in relation to financial and staffing plans. The Financial graph shows the cumulative dollars spent to date compared with the projected budget. The Staffing profile reveals the number of personnel working on the project each month with respect to the planned staffing levels. The initial baseline plan is established at the beginning of the project. On a monthly basis, the actuals are compared to the plan. In the event an activity needs to be replanned, the original plan is retained as a baseline and a Current Operating Plan (COP) is used to forecast the revised spending and staffing. In our sample project, we recognized the need to take corrective action to fix the design problem highlighted in the Target System Resource Usage Report. To accomplish this, more project personnel than originally planned were assigned early in the recovery effort (during months 8-9) to preserve the schedule. Also, during months 17-20, it appeared that project staff were being redirected to other efforts and, thus, were not available to this project. The Staffing metric pointed to this problem and allowed for corrective action.

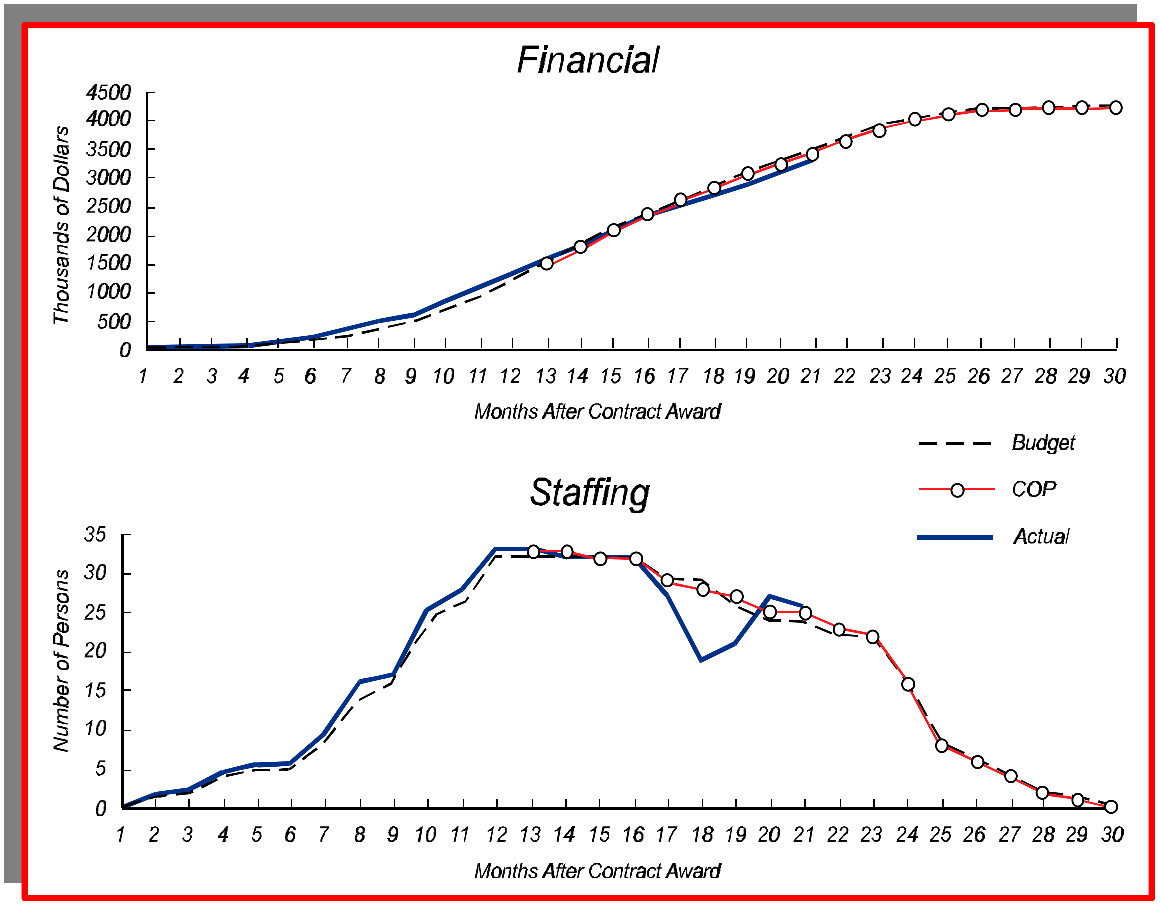


Figure N-4

N.4.6 Size Trend Report

The purpose of the Size Trend Report is to uncover potential development problems related to changes in code size before they become critical. This report is updated monthly to show the estimated size at completion. The size of each Computer Software Configuration Item (CSCI) is tracked via separate Size Trend Reports. The Size Trend Report is used in conjunction with the Productivity Measurement Report. If size is increasing (which is often the case in software development projects) and productivity is not above what was planned, a significant problem exists requiring immediate attention — namely, at a given productivity rate with an increasing job size, it will take longer to finish the job than what was scheduled.

In our sample project, the size of the job (measured in equivalent source lines-of-code or ESLOC, derived from actual source lines-of-code by taking into account the reduced effort for reused and modified code) was reduced because of an efficient design that introduced more functionality in auto code and less in new code. But, we see that the size of the delivered source lines-of-code (DSLOC) ballooned because of the growth in the less efficient auto code, which directly impacted memory use. What we were faced with was a potential "*fit*" problem requiring more memory than allocated. A new design was required, resulting in corrective action that had an immediate impact on productivity and cost performance.

N.4.7 Productivity Measurement Report

The Productivity Measurement Report is used to evaluate a project's performance by measuring software productivity, which is based on equivalent source lines-of-code (ESLOC) produced per staff month and is reported for each Computer Software Configuration Item (CSCI). What the report shows is the planned productivity (based on the estimated size of the CSCI, cumulative planned staffing and planned percent complete) versus actual productivity (based on current estimated or actual code counts of the CSCI, cumulative actual staffing and the actual percent complete). Not only does this support better project management, but it also establishes a baseline for measuring improvement and future cost estimates needed for bidding. In our sample project, we can see that productivity was running high early in the project due to the lower complexity of the job. However, once it was realized that a portion of the design would have to be redone to correct the memory use problem, the rate of productivity slowed down dramatically (while corrective action was taken) and then gradually recovered.

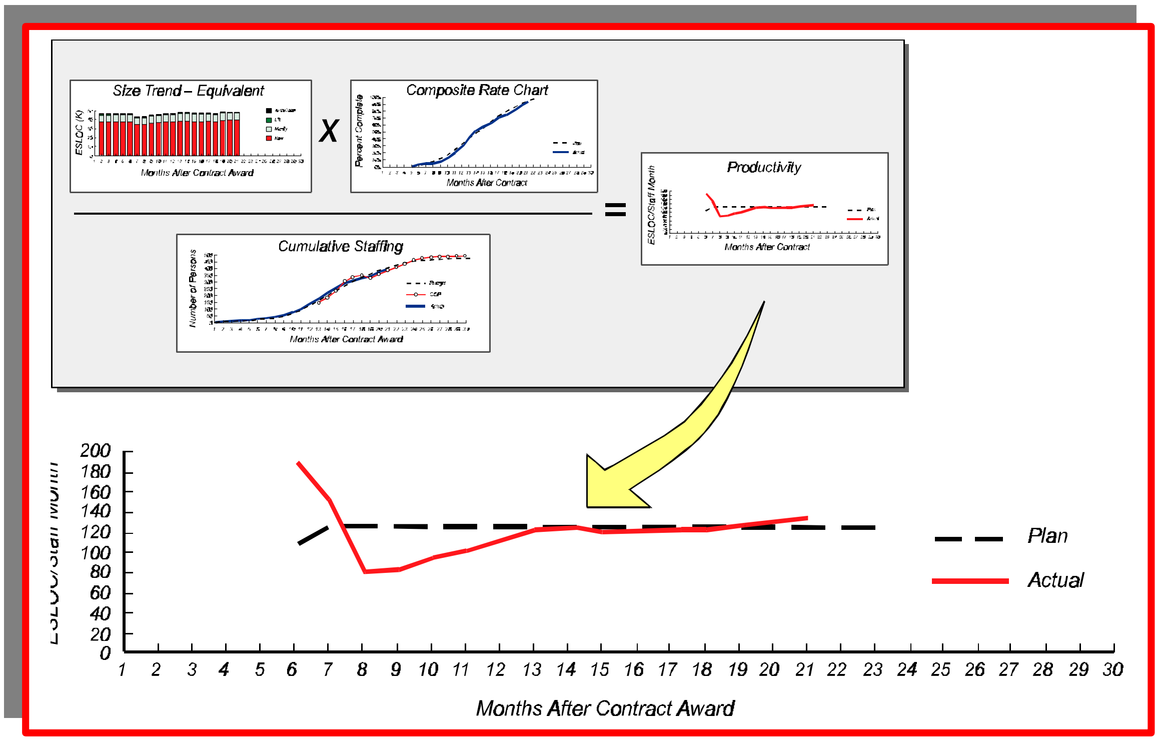


Figure N-5

N.4.8 Software Problem Status Report

The purpose of the Software Problem Status Report is to track the maturity of deliverable products, measuring the status of known software problems that need to be fixed versus the rate of closure of problems. Software problems are tracked after software goes under configuration control, usually at the start of software integration.

The Problem Change Request is a form used to document software problems. Based on the number of opened Problem Change Requests, the Software Problem Status Report serves as an indicator of the maturity of the software product. Software usually is ready for delivery when the rate of finding new problems has decreased significantly and the gap between opened and closed problems is near zero. The following measurements are plotted on the graph:

- Opened shows the number of software problems that have been detected and reported by the designated project authority.
- Resolved shows the number of software problems for which a technical solution has been found and verified.
- Closed shows the number of problems that have had the correction verified and formally closed by incorporating the change in the product baseline.

Some projects also find it advantageous to show the number of software problems a project of this size can expect to encounter over time, based on historical data from similar projects. In our sample project, the problem reporting began in month 16 as integration started. Initially, problems were detected faster than they could be resolved. Without corrective action (for example, assigning more resources), a schedule slip was likely. This project was able to eventually close the gap between opened and closed problem reports and the rate of new problem reports tapered off, indicating that the software was ready for the Build 1 demonstration during month 21.

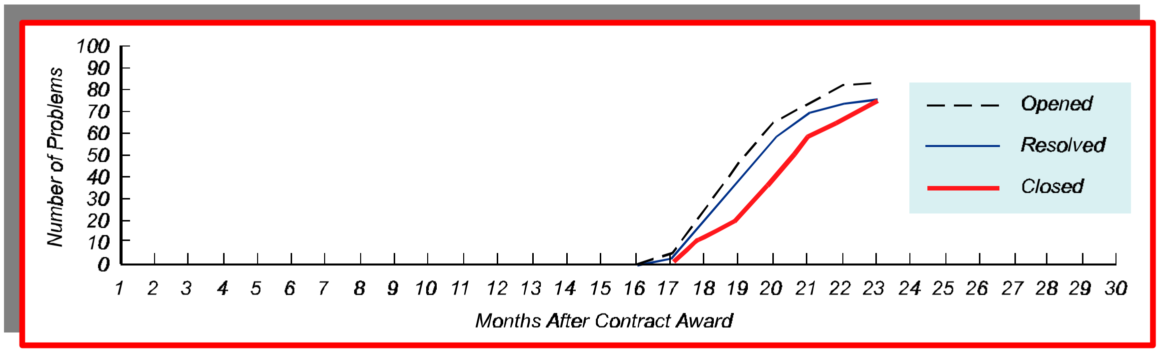


Figure N-6

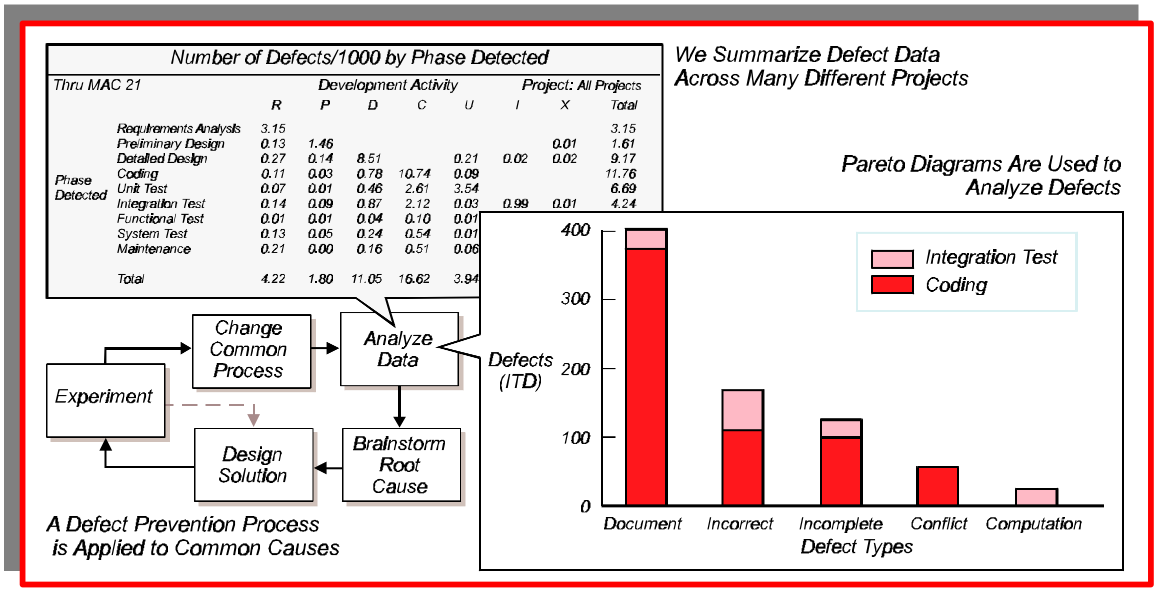
N.4.9 Quality Indicator Reports

The Quality Indicator Reports are summaries of defect data that can be used to understand and improve the software process. Software quality indicators include both product and process defects.

Defects can be introduced into software during each phase of the software development life cycle. Procedures need to be established to prevent these defects, wherever possible, or discover their existence at the earliest possible moment. Data from Hughes projects and other industry sources shows that the earlier you catch the problem, the less costly it is to fix. The Quality Indicator Reports categorize the kinds of defects discovered and the life cycle phase in which the defects were detected. The Quality Indicator Reports consist of the following charts:

- A summary chart showing quality indicator status,
- Charts that focus on type of defect,
- Charts that focus on phase detected, and
- A quality indicator analysis report.

The primary goal of the various charts is to provide management with increased visibility into patterns and trends appearing from the statistical summary of defects discovered during the software life cycle. This data gives the manager and technical personnel the ability to home in on the specific causes of defects. Defect prevention teams can analyze these charts and outline a plan of action to eliminate the root cause of the most frequent and/or costly defects. The Quality Indicator Reports are an increasingly important part of the Hughes approach to continuous measurable improvement (cmi).



Number of Defects/1000 by Phase Detected

Thru MAC 21 — Development Activity — Project: All Projects

Phase Detected	R	P	D	C	U	I	X	Total
Requirements Analysis	3.15							3.15
Preliminary Design	0.13	1.46					0.01	1.61
Detailed Design	0.27	0.14	8.51		0.21	0.02	0.02	9.17
Coding	0.11	0.03	0.78	10.74	0.09			11.76
Unit Test	0.07	0.01	0.46	2.61	3.54			6.69
Integration Test	0.14	0.09	0.87	2.12	0.03	0.99	0.01	4.24
Functional Test	0.01	0.01	0.04	0.10	0.01			
System Test	0.13	0.05	0.24	0.54	0.01			
Maintenance	0.21	0.00	0.16	0.51	0.06			
Total	4.22	1.80	11.05	16.62	3.94			

Figure N-7

N.4.10 Defect Density Tracking Report

Defects are quite simply deficiencies in a product or process (flaws in the design or coding process, for example) that can lead to development of a product that fails to meet customer requirements. They are found during both internal and external reviews and testing of software products under development. Projects identify defects during reviews based on review procedures and checklists. Defects are then documented when identified, including information on the type of defect, the cost to repair, and the phase in hich each defect was detected.

Defects cost time and money. Even with defect prevention techniques, experience shows that the best software processes today are not yet capable of "*defect free*" results. So, we can expect to find defects. Historical data from past Hughes projects indicates that it costs significantly less to find and fix defects in the development phase that caused them rather than later in the project. The early detection of problems is the key to maintaining high productivity at minimum costs.

The Defect Density Tracking Report is used to compare the number of defects being found in each phase of software development with historically derived control limits. The control limits determine a planned range of defects for each phase that a typical project should experience. The actual defect rates being found are then compared with the control limits to provide insight into the process being used on the project.

If the actual defect rates are within the control limits of the planned defect density, the process being used by the project is performing as expected. The fact that defect rates are lower than the lower control limit requires further analysis. It may indicate that insufficient reviews are being conducted, calling for immediate corrective action. Or, this may mean that the project has found a process improvement that has produced fewer defects. If so, the improved process should be documented for others to use. Similarly, defect rates that exceed the upper control limit indicate conditions that should be examined for possible improvements.

In the sample project, where Coding Defect Density was measured per thousand equivalent source lines-of-code (KESLOC), we saw that the actual data was within the control limits showing that the project was performing as expected. Hughes is constantly improving its metrics program. The Defect Density Tracking Report is a recent addition and is currently being piloted on some projects.

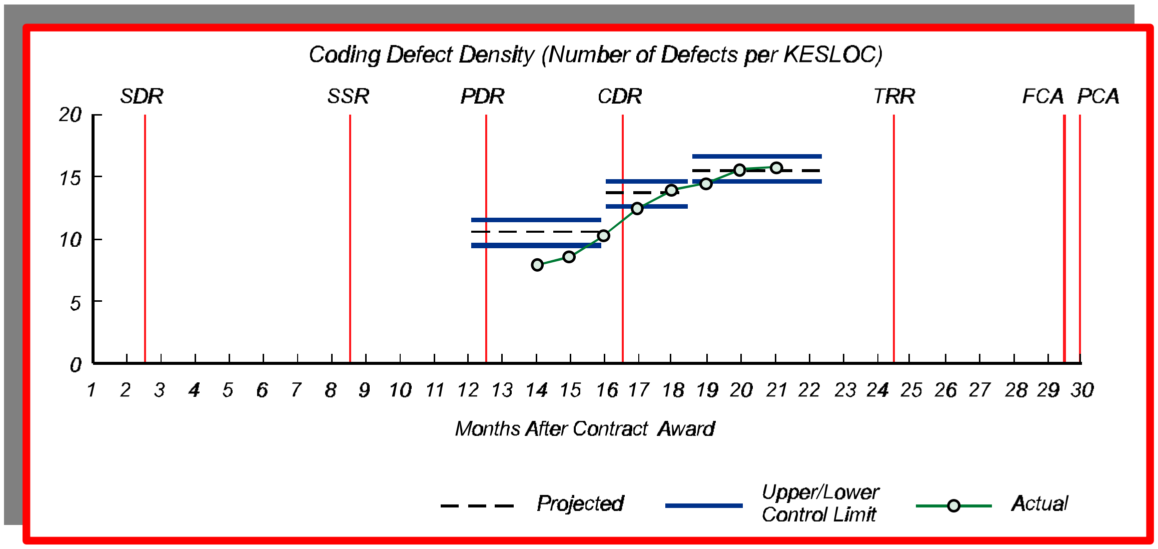


Figure N-8

N.4.11 Target System Resource Usage Report

Target System Resource Usage reporting is mainly concerned with the management of computer resources (e.g., main memory, processor time, mass storage, etc.), measured in terms of percentage utilized. Selection of which resources to monitor is done in the requirements analysis phase based on how critical the resource is or how risky it will be to meet the requirement.

Initially, control limits are established that represent the maximum resource utilization allowed under the contract, along with a management reserve that decreases in size as the ability to estimate/measure the utilization improves during the life of the project. Exceeding the management reserve figure indicates a "*risk*" calling for management action. During the life of the project, the actual resource utilization is estimated with more and more precision until actual measurements finally replace the estimates.

The key to success here (as in all the metrics) is to manage risk early in the project. In our sample project, the Target System Resource Usage Report provided an estimate that showed the new design would require too much memory. This estimate (long before actual memory usage could be measured) allowed us to focus on the problem early enough so that corrective action could be made with minimal effects on the project. Without the use of metrics, the problem would not have surfaced until very late in the project, during the final stages of integration and testing, resulting in a major redesign effort that would have had a severe impact on cost and schedule.

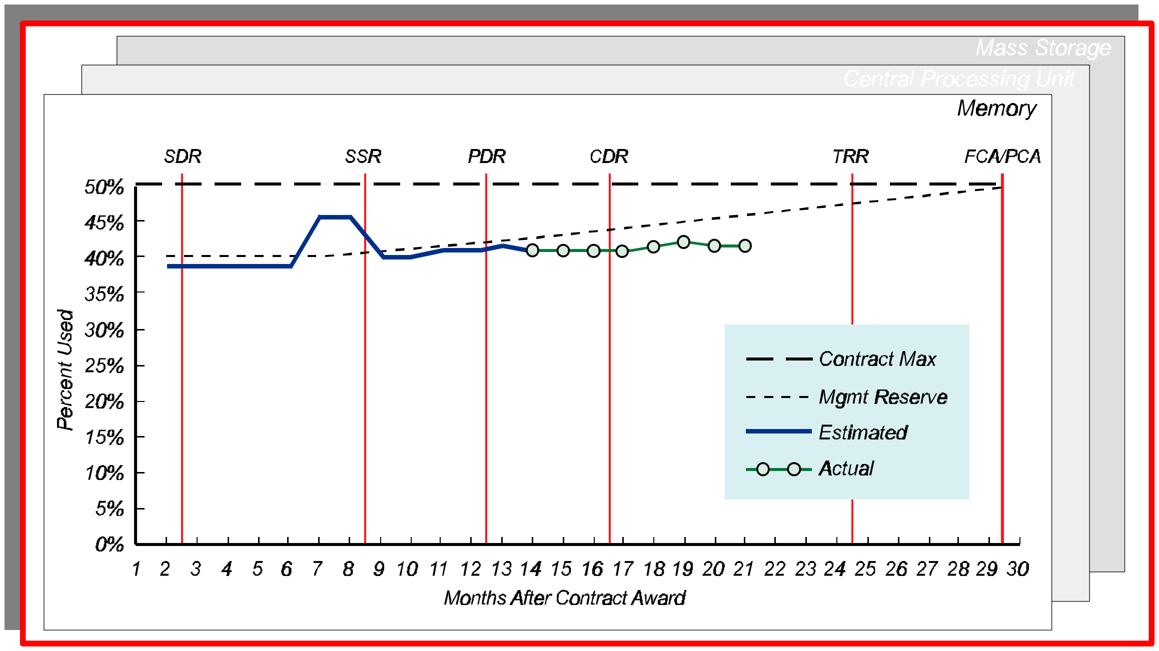
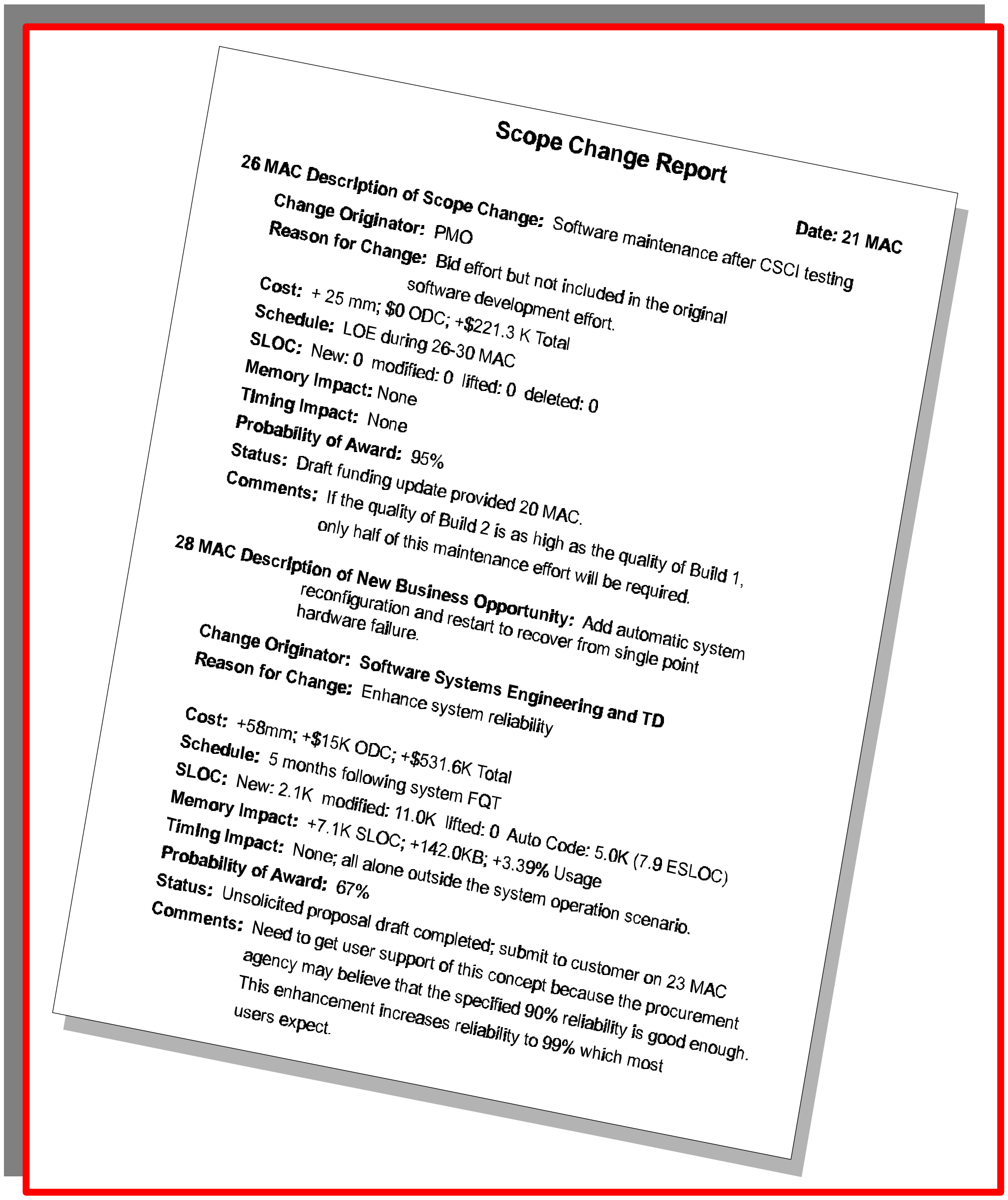


Figure N-9

N.4.12 Scope Change Report

The Scope Change Report describes any addition or deletion in the scope of work. It forces management to "size up" a new task or requirement and assess the impact of these changes on cost and/or schedule. Essentially, it is a technique for managing (and forecasting) changes in the contract's scope of work, common to evolutionary product development, which gives project management a basis for understanding what is currently in the baseline and what is not. This planning tool is also used to look at related new business opportunities for a project (normally follow-on work from existing contracts) and to assess their impact, both technically and financially.



Scope Change Report

Date: 21 MAC

26 MAC Description of Scope Change: Software maintenance after CSCI testing

Change Originator: PMO

Reason for Change: Bid effort but not included in the original software development effort.

Cost: + 25 mm; $0 ODC; +$221.3 K Total

Schedule: LOE during 26-30 MAC

SLOC: New: 0 modified: 0 lifted: 0 deleted: 0

Memory Impact: None

Timing Impact: None

Probability of Award: 95%

Status: Draft funding update provided 20 MAC.

Comments: If the quality of Build 2 is as high as the quality of Build 1, only half of this maintenance effort will be required.

28 MAC Description of New Business Opportunity: Add automatic system reconfiguration and restart to recover from single point hardware failure.

Change Originator: Software Systems Engineering and TD

Reason for Change: Enhance system reliability

Cost: +58mm; +$15K ODC; +$531.6K Total

Schedule: 5 months following system FQT

SLOC: New: 2.1K modified: 11.0K lifted: 0 Auto Code: 5.0K (7.9 ESLOC)

Memory Impact: +7.1K SLOC; +142.0KB; +3.39% Usage

Timing Impact: None; all alone outside the system operation scenario.

Probability of Award: 67%

Status: Unsolicited proposal draft completed; submit to customer on 23 MAC

Comments: Need to get user support of this concept because the procurement agency may believe that the specified 90% reliability is good enough. This enhancement increases reliability to 99% which most users expect.

Figure N-10

N.5 Afterword

When flying a plane from Los Angeles to San Francisco, a pilot knows the destination and the direction in which to begin the flight. During the journey, however, winds may be encountered that cause the plane to go off course. The earlier the pilot detects the problem and corrects for the wind, the more likely the flight will arrive on schedule. Similarly, software metrics provide the software project manager with the information necessary to determine when a software development project is on course. As shown by the sample project in this brochure, metrics analysis enables early detection of problems and allows corrective actions to ensure delivery of quality products that are on time, within budget, and meet customer expectations.

Hughes has developed several tools for in-house use in collecting, analyzing, and reporting software metrics data. These include the Quality Indicator Program (QIP) and the Quantitative Process Management Information System (QPMIS). QIP has been used to collect data on nearly 100,000 defects during the past five years. Defect prevention teams analyze this data for root causes and select pilot projects to test proposed changes. Valid improvements are incorporated into the organization's practices and procedures.

QPMIS captures metrics data in a common database and automates the analysis and reporting for many of the reports described in this brochure. QPMIS allows software project managers to concentrate on the content and interpretation of the data and not concern themselves with formatting reports. A centralized historical database aggregates the information from each completed project and allows electronic access to the data for organization-wide trend analysis and for future reference.

Hughes has made a strong commitment to metrics over the years with significant support from the Software Network Management Council and the Company-wide Software Initiatives Program. Training courses are offered regularly and cover a wide range of topics related to metrics including Software Project Reporting, Introduction to Quantitative Process Management, Defects Collection, Analysis and Prevention, and Reaching for Higher Levels of Software Process Maturity.

Software development is a human intensive activity and, as such, is subject to human error. However, many errors and much costly rework can be avoided by continuously measuring and optimizing defined development processes. Improvements pay off in many ways. Lessons learned from past projects enable new projects to be more efficient. Analysis of historical data results in greater accuracy in predicting costs, schedules, and risks during proposal activities. Even small cost performance index improvements can translate into substantial savings when accumulated over time on multiple projects.

The successful companies of the future will be those that take advantage of metrics to institutionalize "*best in class*" practices and procedures for software development.

N.6 Glossary/Acronym List

ACWP	Actual Cost of Work Performed
BCWP	Budgeted Cost of Work Performed
BCWS	Budgeted Cost of Work Scheduled
CDR	Critical Design Review
cmi	Continuous Measurable Improvement
COP	Current Operating Plan
CPI	Cost Performance Index
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
DSLOC	Delivered Source Lines-of-code
ESLOC	Equivalent Source Lines-of-code
FCA	Functional Configuration Audit
FQT	Formal Qualification Test
IPD	Integrated Product Development
IRS	Interface Requirements Specification
KESLOC	Thousand Equivalent Source Lines-of-code
LOE	Level of Effort
MAC	Month After Contract
ODC	Other Direct Costs
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
PMO	Project (or Program) Management Office
QIP	Quality Indicator Program
QPMIS	Quantitative Process Management Information System
SDD	Software Design Document
SDP	Software Development Plan
SDR	System Design Review
SEI	Software Engineering Institute
SLOC	Source Lines-of-code
SPI	Schedule Performance Index
SRS	Software Requirements Specification
SSR	Software Specification Review
STD	Software Test Descriptions
STP	Software Test Plan
STR	Software Test Report
TD	Technical Director
TRR	Test Readiness Review